future internet

MDPI

Article

Research and Design of a Decentralized Edge-Computing-Assisted LoRa Gateway

Han Gao, Zhangqin Huang *, Xiaobo Zhang and Huapeng Yang

Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, 100 Pingleyuan, Chaoyang District, Beijing 100124, China; gao2009han@yeah.net (H.G.); xiaobo.zhang@emails.bjut.edu.cn (X.Z.); hpyang@emails.bjut.edu.cn (H.Y.)
* Correspondence: zhuang@bjut.edu.cn

Abstract: As a narrowband communication technology, long-range (LoRa) contributes to the long development of Internet of Things (IoT) applications. The LoRa gateway plays an important role in the IoT transport layer, and security and efficiency are the key issues of the current research. In the centralized working model of IoT systems built by traditional LoRa gateways, all the data generated and reported by end devices are processed and stored in cloud servers, which are susceptible to security issues such as data loss and data falsification. Edge computing (EC), as an innovative approach that brings data processing and storage closer to the endpoints, can create a decentralized security infrastructure for LoRa gateway systems, resulting in an EC-assisted IoT working model. Although this paradigm delivers unique features and an improved quality of service (QoS), installing IoT applications at LoRa gateways with limited computing and memory capabilities presents considerable obstacles. This article proposes the design and implementation of an "EC-assisted LoRa gateway" using edge computing. Our proposed latency-aware algorithm (LAA) can greatly improve the reliability of the network system by using a distributed edge computing network technology that can achieve maintenance operations, such as detection, repair, and replacement of failures of edge nodes in the network. Then, an EC-assisted LoRa gateway prototype was developed on an embedded hardware system. Finally, experiments were conducted to evaluate the performance of the proposed EC-assisted LoRa gateway. Compared with the conventional LoRa gateway, the proposed edge intelligent LoRa gateway had 41.1% lower bandwidth utilization and handled more end devices, ensuring system availability and IoT network reliability more effectively.

Keywords: LoRa gateway; edge computing network; internet of things; embedded system; distributed computing

Citation: Gao, H.; Huang, Z.; Zhang, X.; Yang, H. Research and Design of a Decentralized Edge-Computing-Assisted LoRa Gateway. *Future Internet* **2023**, *15*, 194. https://doi.org/10.3390/fi15060194

Academic Editor: Heming Cui

Received: 4 April 2023
Revised: 24 May 2023
Accepted: 26 May 2023
Published: 27 May 2023

1. Introduction

LoRa is one of the low-power wide-area network (LPWAN) technologies. The characteristics of LPWAN include long-range communication, short packets, low power, and low bandwidth [1]. As a result, LoRa can enable energy-efficient connectivity for many widely dispersed end-point IoT devices. In many IoT applications, such as environmental monitoring and smart cities, the LoRa gateway acts as a bridge that transmits messages from the end device to the cloud processing center. As the Internet of Things grew, so did the number of apps that support real-time latency. Due to the huge amount of computing data generated by IoT end applications, effectively reducing the latency of latency-sensitive IoT applications is a huge challenge for LoRa gateways currently. In addition, LoRa gateways provide services for data transfer between end-point IoT devices and cloud computing. This information is subsequently analyzed and saved on the cloud. Consequently, the LoRa gateway's computation and storage capabilities are underutilized, and are subject to certain potential dangers. The difficulties include enhancing the effectiveness and safety of the LoRa system.

With the development of IoT and communication technology, more and more sensor devices are connected to the IoT in various ways, and a large amount of data is generated as a result. This massive amount of data causes several issues, including excessive latency, necessitating a larger bandwidth. Transmission of data between cloud centers and IoT devices will result in significant delays. Edge computing may deliver services at the network's edge to minimize latency and provide edge data processing power and security [2]. Edge computing became popular as a result of the installation of servers at the edge [3]. Edge computing employs a decentralized strategy that allows application processing and task execution to occur locally. In this manner, the network traffic issue is reduced. In edge computing, access to resources is provided at the network's edge [4], close to end devices, which has a significant influence on bandwidth and minimizes communication latency [5]. With the development of edge computing technology, the processing power of traditional cloud computing can be partially migrated to the LoRa gateway. This decentralized strategy may aid in preventing the falsification of data when applied to the LoRa gateway. Consequently, an EC-assisted LoRa gateway is anticipated to enhance both computing performance and security.

The issue of job scheduling in the LoRa gateway using edge computing is an essential subject of debate. A lot of research has focused on the job scheduling of edge computing resources in recent years. Nonetheless, the majority of past initiatives have encountered two significant limitations, namely a lack of practicability and a lack of generalizability. Existing simulation-based systems, for instance, are inapplicable to the majority of real-world circumstances. Most present solutions are appropriate for a limited range of IoT applications but cannot be generalized to compute-intensive and time-sensitive applications. Moreover, several algorithms are engaged in sophisticated offloading procedures that demand a considerable amount of decision time, thereby rendering the schemes unfriendly. The main research contributions of this paper are as follows.

First, we built a LoRa gateway with edge computing on the Zynq SoC platform, which is called the EC-assisted LoRa gateway.

Then, based on the characteristics of the LoRa network, we used the delay awareness algorithm with high-availability edge computing to monitor and maintain the edge node failures through a distributed method. On the one hand, we want a reverse tile gateway that can run locally or together with existing network server systems. In order to build a resource-concentrated and latency-aware IoT application, we have been developing a decentralized method for grouping edge nodes in a distributed pattern.

In addition, in order to improve the network reliability of the Internet of Things in the distributed deployment mode, we have constructed a distributed edge node group for the end nodes in the network, and performed maintenance operations such as detection, repair, and replacement on each node in the group.

Finally, in order to enable the new algorithm LAA to run on the LoRa gateway, we modified the ARM-based embedded Linux operating system in the LoRa gateway to generate an image.ub that is compatible with third-party lib libraries and plugins. The experimental results indicated that the EC-assisted LoRa gateway can achieve the same performance while using fewer resources than a traditional LoRa gateway.

The rest of the paper is organized as follows. Section 2 discusses related work, and Section 3 describes a design for an EC-assisted LoRa gateway. Then, the experiments and their results are elaborated on in Section 4. Section 5 concludes this article.

2. Related Work

For the growth of IoT services, efficiency and security challenges become crucial. A total of 75.44 billion devices will be linked globally by 2025 as a result of the expansion of 5G and the fast development of the IoT, generating vast quantities of data [6]. The fast growth of IoT devices, as well as the massive data traffic generated at the network's edge, imposed new strains on the current centralized cloud computing architecture due to bandwidth and resource constraints. As a possible option, edge computing has garnered

considerable interest. Furthermore, edge computing is closer to the edge side of IoT than cloud computing, which can provide computing services close to the end devices, and therefore can provide data pre-processing and part of machine learning model training services [7]. Incorporating edge computing into the LoRa gateway remains conceptually and technically problematic owing to the diverse needs and limited resources of IoT devices.

2.1. LoRa Gateway and System

As described in Section 1, LoRa communication exhibits distinctive characteristics, such as simple and cost-effective networking, robust interference resistance, and unparalleled advantages in large-scale low-power networking and IoT services in remote areas, where cellular base stations are inaccessible. The fundamental structure of a LoRa communication system is illustrated in Figure 1, where end devices transmit data packets to gateways, which in turn route them to network servers via either cellular networks or Ethernet. The focus of this paper is investigating how to leverage distributed computing to maximize the gateway capabilities at the edge for data processing, and thereby alleviate the burden on cloud computing centers. The hardware design of both the terminal devices and the LoRa gateway can be customized to accommodate different types of sensor interfaces and service requirements.

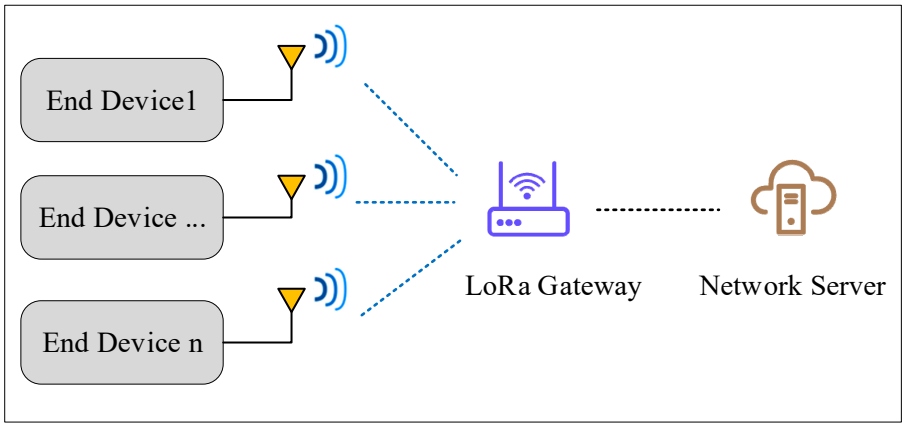

Figure 1. LoRa gateway system.

At the LoRa communication level, several implementations of LoRa systems were proposed in [8], including XisLoRa, Chirp Stack, and the Things Network (TTN). The communication architecture design proposed in [8], which was targeted at different applications for uplink and downlink, has provided inspiration for the design approach in this article.

In the field of distributed computing, Danish et al. [9] and Lin et al. [10] both proposed applying the technology of the blockchain to LoRa networks. However, their proposed approach was to deploy blockchain technology in the cloud, which would not be effective in reducing the load pressure on the cloud, but would rather increase the processing workload of the cloud. The LoRa gateway's edge computing capabilities would not be activated. Liu et al. [11] proposed a LoRa system based on edge computing which migrates some of the access control command instructions from the cloud server to the LoRa gateway, thus effectively relieving the burden of cloud processing. Ozyilmaz and Yurdakul [12] proposed a LoRa system based on blockchain technology. This system would give different devices with different computing and storage capacities different ways to manage their data. High-capability LoRa gateways can download the whole blockchain, but low-capability LoRa gateways can only obtain block headers.

Regarding the distributed applications of LoRa, most of the existing literature (e.g., [13,14]) focuses on utilizing deep learning techniques for processing the collected data on LoRa gateways and end devices, with applications in areas such as fall detection and agriculture. However, there is relatively little research on the massive uplink end device access and

multi-task downlink publication to end devices using LoRa. LoRa gateways typically only provide data relay services between end devices and cloud services, with limited utilization of resources beyond their communication functions. To use edge computing in LoRa gateways, the system architecture of LoRa gateways must be refactored, and the computing resources must be evaluated.

2.2. Edge Computing Network

To support creative IoT applications for edge devices and allow the success of the EC-assisted IoT paradigm, the academic community and industry have suggested a vast array of EC designs and technologies. In this category are cloudlets (small servers), vehicular (or portable) EC (VEC), and edge cloud. These technologies primarily aid in the deployment of applications in challenging environments with rapid temporal variation. In addition, there are mobile edge computing (MEC) and mobile cloud computing (MCC) technologies that enable the implementation of extensive computing applications on local IoT smart devices. This is achieved by offloading a significant percentage of programs to the devices themselves.

Due to the diverse nature of the resources accessible to edge nodes, IoT applications confront several obstacles, including the need for flexibility, low latency, high bandwidth, error-handling capabilities, and capacity. Computing at the network's edge offers adaptive resources that allow distributed computing and safeguard data from errors typical of a centralized system. Some research, such as on cloudlets [15], femtoclouds [16], and edge computing, has emphasized the incorporation of mobile device resources. When a device detects edge support, it transfers the majority of its processing to a cloudlet rather than building task-specific components [17]. The concept of cluster computing in femtoclouds necessitates centralized management by expert controllers [18]. By distributing specialized servers to satisfy end users' needs in a particular place, edge computing makes this feasible. The mechanisms for clustering-based methods are detailed in [19–21].

The previously mentioned studies have tended toward a centralized strategy for the structure and administration of different resources, such as operating systems and applications [22]. To meet deadlines on time, it is necessary to use a decentralized method. This decentralized strategy delivers resources at the network's edge [23].

When an IoT application is operating on a collection of edge networks, it is crucial for the edge computing to be dependable and fault-tolerant [24]. Due to the vast range of edge devices, networks, and data processing methodologies, it is a significant challenge to create reliable network services and effective fault-tolerant solutions in edge computing networks. Refs. [25–27] have conducted research on methods for fault detection and correction in edge nodes.

As we have seen, many researchers have attempted to deploy distributed computing methods in LoRa gateways and use various techniques to address the edge computing node failure problem, but these methods typically use centralized control for error handling, which often results in significant resource waste and communication delays [26]. In contrast to previous studies, the main significance of this study lies in leveraging LoRa gateways to share the distributed computing load, and in providing an efficient method for accessing and managing large-scale LoRa nodes as distributed edge nodes, which effectively reduces the burden on cloud computing processing and system energy consumption. As a result, this study provides a good edge computing approach for wireless IoT systems that utilize large-scale edge sensing LoRa nodes in applications such as smart forestry and agriculture.

3. Design of the EC-Assisted LoRa Gateway

3.1. LoRa Gateway Hardware System Design Based on Zynq SoC

The LoRa gateway hardware system is based on a XINLIX Zynq7000 processor. Zynq SoC architecture combines the software programmability of a processor with the hardware programmability to form a dual-core ARM Cortex-A9 processor and a conventional FPGA logic component, providing unparalleled system performance, flexibility, and scalability.

The communication module of the LoRa gateway uses the LoRa SX1278 baseband chip, which features a small size, low power consumption, long transmission distance, and high interference immunity. The LoRa gateway can receive, process, store, and forward the data of end devices. By assigning LoRa gateway edge computing capability, it can effectively enhance data processing capability and reduce data transmission delays. The proposed EC-assisted LoRa gateway hardware system architecture is shown in Figure 2.

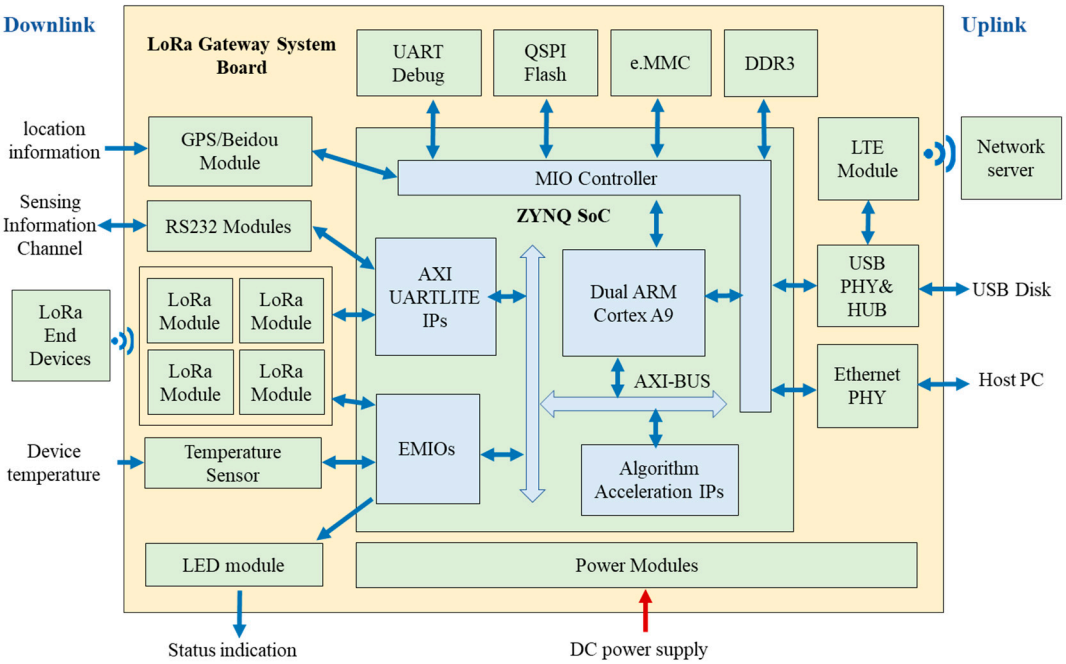

Figure 2. LoRa gateway hardware system based on Zynq SoC.

As shown in Figure 2, the main modules connected to the PS resource include the sensor interface, the system debugging interface, the storage module, the communication module, the Ethernet interface, etc. The storage module mainly includes QSPI FLASH, DDR3, and eMMC, which are used to store the running memory, system files, application files, and data files. The wireless communication method for uplink in the system is via 3G, 4G, or LTE modules, which are connected to the network server, while the wireless communication for downlink relies on four LoRa modules, which are connected to numerous end devices. The communication interface between the LTE module and the Zynq is USB, and the communication interface between the LoRa module and the Zynq is UART pass-through. Furthermore, the system also includes interfaces for sensor information acquisition and debugging. In order to connect to multiple serial devices, we utilized multiple AXI_UARLITE IP core ports in our system. We developed the software for multi-threaded design, communication interface development, and distributed computing using the Xilinx Software Development Kit (SDK) environment. The program was compiled using appropriate tools to generate an executable file that was subsequently executed on the Zynq platform.

Algorithmic IP cores for data processing have been integrated into the PL side based on edge requirements to further improve the efficiency of edge computing. However, the end device access and management calculations in this article were run on the dual-core ARM Cortex-A9 on the PS side.

3.2. Edge Computing in the LoRa Gateway

Using the LoRa gateway with edge computing capabilities, two modules were transferred from the network server to the LoRa gateway. Through network control (NC), which processes application packet data, and JS, which manages the connection operations of the

end devices using connection operations, the LoRa gateway generated and stored contextual data for the end devices. Conversely, the LoRa gateway required certain contextual data to perform application packet processing tasks.

As shown in Figure 3, when the LoRa gateway receives a network request from an end device, it first verifies the legitimacy of the request. Then, the LoRa gateway creates session data for the end device, including DevAddr, two session keys, and some metadata. The LoRa gateway requests and creates a new block with a transaction containing these data. In the meantime, the LoRa gateway must produce and transmit a join accept message to the end device. The end device's successful acceptance of the join accept message shows that the join operation was successful. It may be sent from the network server to the LoRa gateway because the NC module is placed in the LoRa gateway, which is utilized for application data processing.

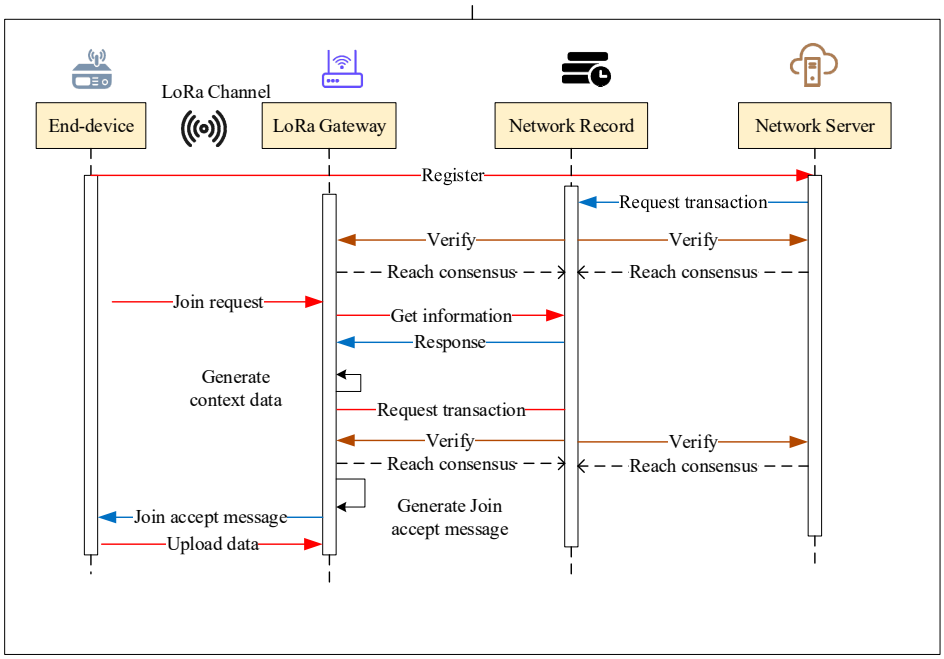

Figure 3. The EC-assisted LoRa Gateway workflow of data.

Uplink Data Processing: When an end device reports application data, the NC processing module in the LoRa gateway divides the application into three parts: metadata, encrypted data, and the message integrity code (MIC) value. Then, the NC module checks the integrity of the application data by extracting the DevAddr field from the metadata and querying the session data of the end device to verify the authenticity of the MIC. If the MIC value of the application packet does not match, the packet is rejected. After the verification is passed, the LoRa gateway transmits the encrypted data applied by the end device to the network server. At the same time, the LoRa gateway must send an acknowledgment message (ACK) to the end device, confirming that the application packet has been successfully received.

Downlink Data Processing: When sending data to the end device, the network server first creates a new connection and encrypts the application data sent down. The encrypted data are then sent from the network server to the LoRa gateway. The LoRa gateway contains the downlink device application data packets, and the MIC value information is generated and calculated by the LoRa gateway, which is encapsulated in the NC module. When the end device reports data, the LoRa gateway obtains the target information of the end device, while the downlink application packets are sent to the end device through the wireless channel of the LoRa module.

LoRa gateways that are deployed on near-end devices can improve the efficiency of cloud computing processing in the system by means of edge computing. Among them, the

LoRa gateway mainly handles the parsing and encapsulation of application packets from end devices and MIC calculations, which are no longer handled by the central cloud server. On the one hand, the computing resources in the LoRa gateway can be fully utilized to relieve the computing pressure in the cloud server, and at the same time, the LoRa gateway can perform pre-processing operations on the application data reported by the end devices, such as the verification of MIC, without affecting the processing of the cloud server or the transmission function of the application data.

3.3. Distributed Task Execution Mode in the LoRa Gateway

The symbols applied in this paper are shown in Table 1.

Table 1. Symbols applied in this article.

Symbol	Description
EN_k	The LoRa gateway tasks to be processed
i_k	The LoRa gateway receives the task transmitted by the ith edge node
u_k	The LoRa gateway presently allocated tasks
u_{car}	The LoRa gateway task load at time t
u_{need}	The LoRa gateway needed task load at time t
a_k	Size of task-related data
e_k	Processing deadline for task
$t_{e\delta_i}$	CPU time required for a subtask to be completed at edge node e
δ_n	The nth subtask
N_i	Instructions required to complete a subtask
δ_p	Execution rate of edge nodes
Y	Amount of edge nodes executing subtasks
REN_k	The resources presently allocated for task EN_k at time t
$\sum R$	The total resources allotted to the task at time t
DEN_k	The current resource requirement of the task at time t
SEN_k	Task share all available resources
E	The group of nodes connected to the edge network
Rrs	Requirements for the level of reliability between nodes and their neighbors
$G_i^e(EN)$	The task in the ith end nodes group that need to be completed are EN
E^o	Organizer of edge nodes
Q	Network of edge nodes
G	Edge node group
$\sum EN$	The resource designed to handle EN
$\sum ae_i$	The ith edge node requires processing resources
T	The full processing time required to complete all subtasks δ_i
t_{EN}	The time needed to complete task EN in processing

We set the characteristics of $EN_k = \{i_k, u_k, a_k, e_k\}$, where i_k denotes the ith edge node with the execution request. u_k denotes the workload, a_k denotes the size of task-related data, and e_k denotes the deadline of the task. The task $EN_k = \{\delta_1, \delta_2 \ldots\ldots \delta_n\}$, without any limitations. These subtasks may be undertaken concurrently on several edge nodes without a specific order. In Table 1, the significant equation variables are shown. The required resources are provided in (e_k) time to complete the task faster than its objective, in order to process an edge node network subtask and ensure a fair procedure. For the sake of this paper, we will consider each task's required CPU time as a resource. For one subtask of the task EN_k, Ci millions of instructions (MI) were required. The following subtasks δ_i will be loaded.

$$u\delta_i = \sum_{i=1}^{n} C_i \tag{1}$$

On the basis of speed δ_p, the CPU time ($t_{e\delta_i}$) required for processing the EN_i on the edge of the network e is shown below.

$$t_{e\delta_i} = \frac{u\delta_i}{\delta_p} \tag{2}$$

For the EN_k consisting of n subtasks, the LoRa gateway processes the tasks on Y edge nodes in a distributed manner. $REN_k(t)$ denotes the resources (CPU time) required to process the tasks at the edge nodes at time t. $\sum REN(t)$ will be:

$$\sum REN(t) = \int_0^t REN_k(t)dt \tag{3}$$

U_k denotes the resources required to complete all the tasks of a single edge node, U_R denotes the resources allocated to the tasks processed by the edge node at the current moment t, and the remaining required resources U_{need} are denoted as follows.

$$U_{need} = U_k - U_R \tag{4}$$

The current resource demand was given by $DEN(t)$ at time t. For the kth EN_k at time t, $SEN(t)$ describes all resources that are currently accessible. The resources for EN_k employed in the edge network distribution process associated with the cloud are defined as the degree of fairness $fdENk(t)$ of kth EN_k at time t.

$$fdEN_k(t) = \frac{\int_0^t REN_k(t)dt}{\int_0^t \max\{DEN_k(t), SEN_k(t)\}dt} \tag{5}$$

The degree of network fairness $fdEN_k(t) = 1$ indicates that all needed network resources have been allotted to perform the kth EN_k at time t. Less than 1 in disparity $fdEN_k(t)$ indicates an unfair execution.

For competent administration, we propose that significant jobs at the edge be spread over a group $G_i^e(EN)$, where i denotes the number of groups and e denotes edge nodes. EN is associated with a group job that has to be executed. As an organizer, E^o is a representation of an edge node that will receive and process the EN_k. The edge node of the organizer will interact with other edge nodes in the group to complete the assigned task and provide output to the end nodes.

In the E^o edge node group, each edge node acquires the available LoRa gateway resources based on the task. Our proposed latency-aware algorithm (LAA) executes in a decentralized way. An edge node joins the edge node group when it accepts handling a task, so that the action may be completed in distribution mode. To enable distributed processing of the E^o, the E^o node will broadcast to the edge environment Q to establish a $G_i^e(EN)$.

Our proposed LAA algorithm is shown in Algorithm 1. First, in steps 3–7, The E^o will broadcast a grouping message for adding edge nodes to the pair queue Q. After the grouping is completed, in steps 9–15, the LoRa gateway will perform tasks on the edge node elements added to the queue Q. Then, in steps 18–21, the nearby edge nodes in the grouping are monitored and are included in the group when the nearby nodes are available. Throughout the process, the task processing is cyclically performed on all edge nodes that join the group.

Algorithm 1. Pseudocode of the LAA algorithm.

Algorithm LAA: Distributed Latency-Aware Task Processing Algorithm

Input: E^o, $G(EN)$, Q, Q_{free}, $\sum EN$, $\sum ae_i$;
Output: $G_i^e(EN)$

for $E^o \in Q$ **do**
  $Q_{free} \leftarrow E^o$;
  E^o send a request to join $u(u \in Q)$;
  E^{orz} response neighborhood edge node $u'(u' \in Q)$
  for $u' \in Q$ **do**
    $Q_{free} \leftarrow u$
  end for
  while $fdEN_k \neq 1$ **do**
    if $Q_{free} \neq \phi$ **then**
      for $e_i \in Q_{free}$ **do**
        if $\sum EN \cap \sum ae_i \neq \phi$ **then**
        e_i pull δ_i
        $\sum EN = \sum EN - \sum ae_i$
        $G_i^e(EN) \leftarrow e_i$
        end if
      end for
    else
      for $e_i \in G(EN)$ **do**
        if $(e_n \in e_i) \cap G(EN) == \phi$ **then**
          e_i send Rrs to e_n
          $G(EN) \leftarrow e_n$
        end if
      end for
    end if
  end while
  $G_i^e(EN) \leftarrow G(EN)$
end for
return $G(EN)$

4. Experiments

4.1. Experimental Setup

The LoRa gateway was built on an FPGA-based embedded hardware system using the industrially specialized Xilinx Zynq CPU board and Cortex-A9 SoC. With this gear, edge computing is feasible. In the central cloud, our system comprised four LoRa gateways and two network servers. Many customizable client applications were deployed on the server that could make direct connections to the LoRa gateway. The end devices were deployed for experimental testing using the SX1278-based LoRa communication module. The end device can typically connect multiple sensors for data collection and upload the collected data to the LoRa gateway.

In Experiment 1 (test join requests from end devices), 100–2200 simulated end devices submitted join requests continually to the LoRa gateway. Each LoRa gateway controlled between 25 and 550 end devices. Each end device operated at random intervals ranging between 10 min and 2 h. If the end device received a join request and accepted the message, the join operation was deemed successful. After 300 s, if no application had been received, the join request failed. The registration messages for end devices are usually very short, and a large number of device accesses can create pressure on the edge computing system in terms of registration information parsing.

In Experiment 1, the evaluation metric used to reflect the real-time performance of the gateway during the registration of a large number of devices was the time it takes for the end node to register with the network service.

Experiment 2 (test application data upload of end devices) emulated between 100 and 2000 end devices. Each LoRa gateway could support between 25 and 400 end devices. All

steps for end device assembly were completed. Therefore, the end devices could perform data uploading normally. Each end device was able to select a time interval of 15–20 s for continuous data reporting. The network server sent an ACK to each package to signal that the uplink data were received properly. If the end device did not receive an ACK within 30 s after the upload, the transmission of the application data package was deemed unsuccessful. After the time interval or a failure, the endpoint continued to transmit. A large number of end device data packets were transmitted with a full load (payload 255 bytes), and by adopting the design proposed in this paper, the performance of the edge gateway was fully utilized.

Experimental metric 2 utilized two performance indicators: system throughput and CPU utilization. System throughput refers to the number of successfully transmitted data in a unit of time by network devices or ports (the maximum data rate that devices can receive and forward without dropping frames). It is an actual value used to measure network performance. CPU utilization is the percentage of CPU resources occupied by running programs, representing the status of the machine running programs at a specific time. Throughput and CPU utilization at their limits can assess whether the utilization of a LoRa network's overall processing and forwarding capabilities has reached its maximum, and whether the fairness of network transmission is ensured.

Experiment 3 (testing the EC-assisted LoRa gateway versus the traditional LoRa gateway) simulated the operation of 100–2200 end devices. During the experiment, four LoRa gateways were deployed to provide data processing services for the end devices. Each LoRa gateway could interact with the end devices in the coverage area and obtain the application information reported by the end devices. Then, a comparative experiment was carried out using the conventional LoRa system technique. In a traditional LoRa gateway system, the JS and NC modules are deployed in a central cloud implementation. As seen in Figure 4, the central cloud must thus handle all application packages. Our comparison demonstrated the enhanced performance of the LoRa gateway with EC-assisted support.

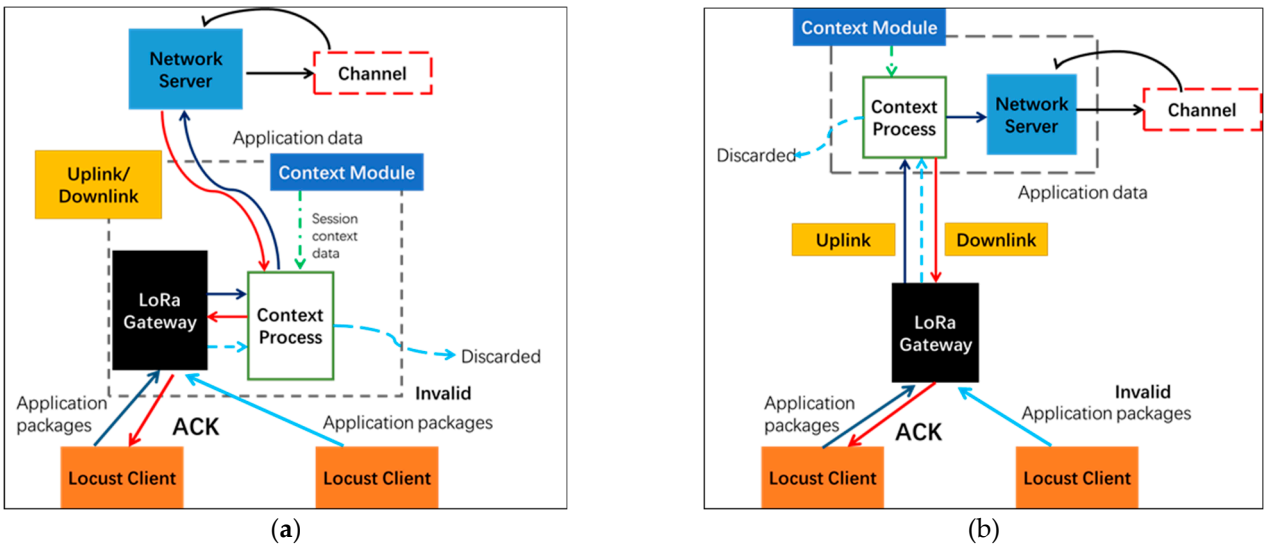

Figure 4. Illustrations of the experiments on the LoRa Gateway with edge computing: (**a**) EC-assisted LoRa gateway system; (**b**) traditional LoRa gateway system.

In Experiment 3, we evaluated the overall network processing and forwarding capability of the LoRa gateway using the metric of bandwidth occupation, which represents the efficiency of receiving and sending messages on the gateway's bandwidth per second.

In conclusion, the evaluation encompassed the gateway's capacity for accommodating large-scale LoRa nodes and processing their uplink data, and the enhancement in bandwidth utilization achieved by EC-assisted LoRa gateways compared to conventional

gateways. Through these experiments, the beneficial effects of the proposed approach were assessed.

4.2. Results and Analysis

Figure 5 provides statistical information on the processing time for access requests. The black dashed line represents the maximum acceptable delay for an end device to receive and accept the message. It may be adjusted to meet certain needs. The join request was deemed unsuccessful if the time barrier was exceeded. Once the number of end devices reached 2000, the mean latency began to exceed the barrier. According to the findings, nearly 75% of end devices could effectively receive and accept messages with a 5 s constraint and 1800 end devices.

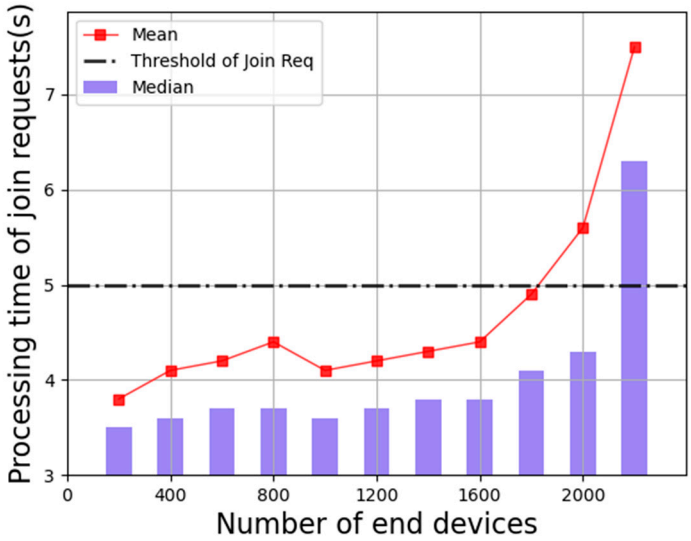

Figure 5. Processing time of join requests with the number of end devices.

Figure 6 depicts the system throughput and CPU utilization of the LoRa gateway and network server while processing application packets from the end devices. The throughput continued to increase until the number of end devices reached 1200, due to underutilized resources. At the same time, CPU usage increased. Once the number of end devices approached 1200, the system throughput stabilized and decreased slightly. This was because as the number of end devices continued to increase, the system became overloaded and the consumption of various resource switches caused a degradation in system performance, thus affecting system throughput. Similarly, the variation in CPU utilization reflected this. The NC module in the LoRa gateway consumed about seven CPU cores. The rest of the cores were occupied by modules used for the operating system and signal processing. However, the processing burden on the network server was alleviated, as the packet processing was now handled by the LoRa gateway with edge computing capabilities.

As the number of end devices increases, the LoRa gateway and network server demand increasing CPU resources. As a result of all packages and invalid packages being processed and checked by the LoRa gateway, the network server's CPU use was decreased in comparison to the prior LoRa system, in which the network server was responsible for handling all packages. As demonstrated in Figure 7, in addition to conserving CPU resources, the bandwidth of the transmission link between the LoRa gateway and a network server was also preserved. Compared to the conventional LoRa system, 1000 end devices required about 41.1% less bandwidth. As the number of end devices increased, the gap continued to expand linearly. One of the advantages of conserving resources is that the saved resources may be used for more genuine packets, protecting them from invalid or malicious traffic. We built edge intelligence algorithms in the LoRa gateway, which were able to validate the effectiveness of end devices and perform specific data processing,

effectively reducing the transmission bandwidth of the LoRa network and reducing the energy consumption.

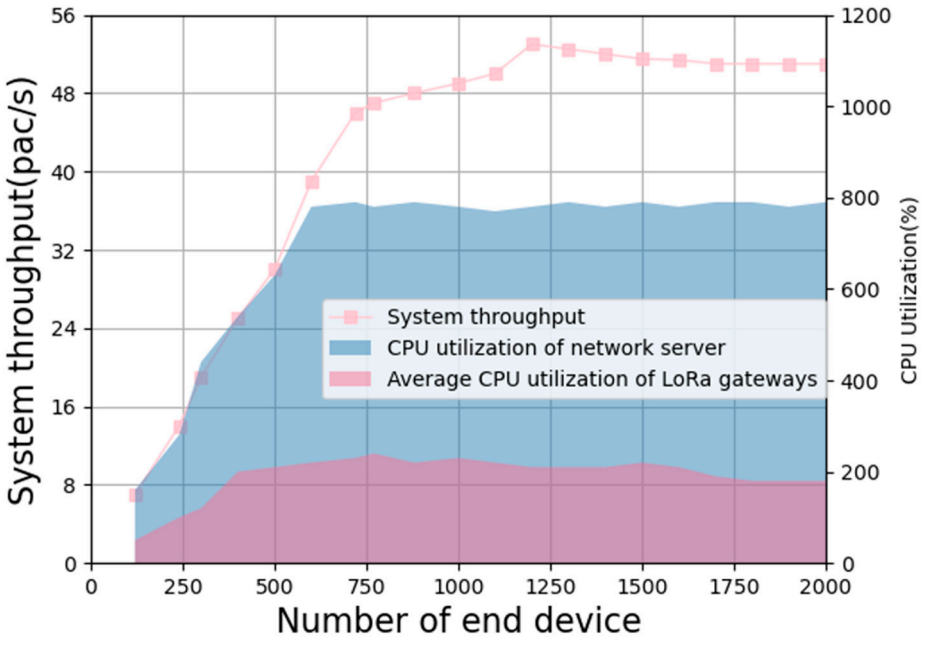

Figure 6. System throughput and CPU utilization.

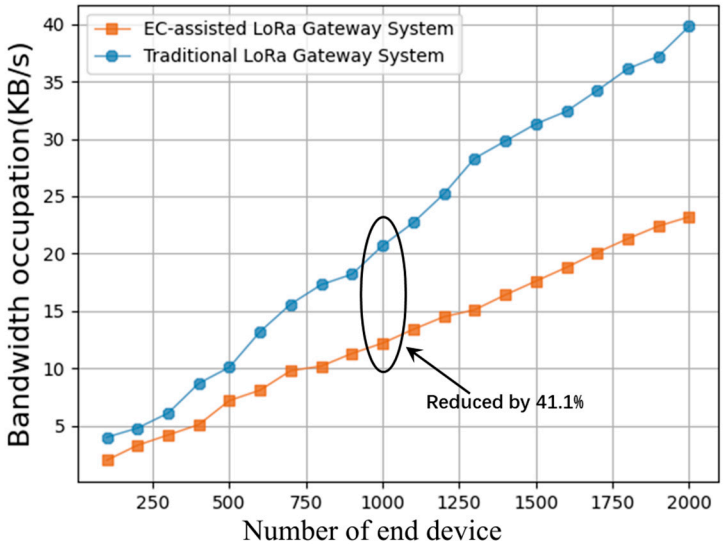

Figure 7. Bandwidth occupation of the EC-assisted LoRa gateway system and traditional LoRa gateway system.

5. Conclusions

This article proposed the design and implementation of a LoRa gateway with edge computing, which is called an "EC-assisted LoRa Gateway". Then, an EC-assisted LoRa gateway prototype was developed on an embedded hardware system, and we proposed a new distributed computing model and used the latency-aware algorithm (LAA) for task processing at the edge nodes in the system, which effectively reduced the latency of task processing and improved the reliability and stability of the network. Experiments demonstrated the efficacy of our approaches, with the suggested LAA algorithm achieving the greatest edge node improvement and performance. We showed the highest performance that the EC-assisted LoRa gateway is capable of achieving: compared to a conventional

LoRa system, the EC-assisted LoRa gateway reduces CPU usage and bandwidth use without compromising system throughput.

In our future work, we plan to further advance research in edge computing. The current proposal did not address issues related to energy efficiency or the detection and repair of failures in edge nodes. In the future, these issues may be considered for achieving high network availability in the context of IoT ecosystems. Furthermore, the experiments presented in this paper were conducted using a simulated LoRa transmission interface, time, and rate, and not with actual massive LoRa end devices. Therefore, future experiments will focus more on the deployment of actual massive wireless modules and data testing to extend the research to a wider range of wireless transmission scenarios.

Author Contributions: Conceptualization, X.Z.; methodology, X.Z. and H.G.; software, X.Z.; validation, H.G. and H.Y.; investigation, H.G. and X.Z.; resources, H.G.; writing—original draft preparation, H.G. and H.Y.; writing—review and editing, X.Z. and Z.H.; visualization, H.Y. and X.Z.; supervision, Z.H.; project administration, Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Raza, U.; Kulkarni, P.; Sooriyabandara, M. Low power wide area networks: An overview. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 855–873. [CrossRef]
2. Gong, C.; Lin, F.; Gong, X.; Lu, Y. Intelligent cooperative edge computing in internet of things. *IEEE Internet Things J.* **2020**, *7*, 9372–9382. [CrossRef]
3. Cui, L.; Yang, S.; Chen, Z.; Pan, Y.; Ming, Z.; Xu, M. A decentralized and trusted edge computing platform for Internet of Things. *IEEE Internet Things J.* **2019**, *7*, 3910–3922. [CrossRef]
4. Bukhsh, M.; Abdullah, S.; Bajwa, I.S. A Decentralized Edge Computing Latency-Aware Task Management Method with High Availability for IoT Applications. *IEEE Access* **2021**, *9*, 138994–139008. [CrossRef]
5. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
6. Latre, S.; Leroux, P.; Coenen, T.; Braem, B.; Ballon, P.; Demeester, P. City of things: An integrated and multi-technology testbed for IoT smart city experiments. In Proceedings of the 2016 IEEE International Smart Cities Conference (ISC2), Trento, Italy, 12–15 September 2016; pp. 1–8.
7. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2015**, *24*, 2795–2808. [CrossRef]
8. Sornin, N.; Yegin, A. *LoRaWAN Backend Interfaces 1.0 Specification*; Lora Alliance Standard Specification: Fremont, CA, USA, 2017; p. 11.
9. Danish, S.M.; Lestas, M.; Asif, W.; Qureshi, H.K.; Rajarajan, M. A lightweight blockchain based two factor authentication mechanism for LoRaWAN join procedure. In Proceedings of the 2019 IEEE International Conference on Communications Workshops (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
10. Lin, J.; Shen, Z.; Miao, C.; Liu, S. Using blockchain to build trusted LoRaWAN sharing server. *Int. J. Crowd Sci.* **2017**, *1*, 270–280. [CrossRef]
11. Liu, Z.; Zhou, Q.; Hou, L.; Xu, R.; Zheng, K. Design and implementation on a LoRa system with edge computing. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference (WCNC), Seoul, Republic of Korea, 25–28 May 2020; pp. 1–6.
12. Ozyilmaz, K.R.; Yurdakul, A. Designing a blockchain-based IoT with ethereum, swarm, and LoRa: The software solution to create high availability with minimal security risks. *IEEE Consum. Electron. Mag.* **2019**, *8*, 28–34. [CrossRef]
13. Queralta, J.P.; Gia, T.N.; Tenhunen, H.; Westerlund, T. Edge-AI in LoRa-based health monitoring: Fall detection system with fog computing and LSTM recurrent neural networks. In Proceedings of the 2019 42nd international conference on telecommunications and signal processing (TSP), Budapest, Hungary, 1–3 July 2019; pp. 601–604.
14. Gia, T.N.; Qingqing, L.; Queralta, J.P.; Zou, Z.; Tenhunen, H.; Westerlund, T. Edge AI in smart farming IoT: CNNs at the edge and fog computing with LoRa. In Proceedings of the 2019 IEEE AFRICON, Accra, Ghana, 25–27 September 2019; pp. 1–6.
15. Yetgin, H.; Cheung, K.T.K.; El-Hajjar, M.; Hanzo, L.H. A survey of network lifetime maximization techniques in wireless sensor networks. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 828–854. [CrossRef]

16. Moridi, E.; Haghparast, M.; Hosseinzadeh, M.; Jassbi, S.J. Fault management frameworks in wireless sensor networks: A survey. *Comput. Commun.* **2020**, *155*, 205–226. [CrossRef]
17. Elsayed, W.M.; Sabbeh, S.F.; Riad, A.M. A distributed fault tolerance mechanism for self-maintenance of clusters in wireless sensor networks. *Arab. J. Sci. Eng.* **2018**, *43*, 6891–6907. [CrossRef]
18. Jamjoom, M.M. EEBFTC: Extended energy balanced with fault tolerance capability protocol for WSN. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 253–258.
19. Scazzoli, D.; Kumar, A.; Sharma, N.; Magarini, M.; Verticale, G. Fault recovery in time-synchronized mission critical ZigBee-based wireless sensor networks. *Int. J. Wirel. Inf. Netw.* **2017**, *24*, 268–277. [CrossRef]
20. Guo, H.; Ren, J.; Zhang, D.; Zhang, Y.; Hu, J. A scalable and manageable IoT architecture based on transparent computing. *J. Parallel Distrib. Comput.* **2018**, *118*, 5–13. [CrossRef]
21. Paul, R.; Melchior, J.; Van Roy, P.; Vlassov, V. Designing distributed applications using a phase-aware, reversible system. In Proceedings of the 2017 1st IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, USA, 25–30 June 2017; pp. 55–64.
22. Hossain, M.S.; Islam, F.B.; Nwakanma, C.I.; Lee, J.M.; Kim, D.S. Decentralized latency-aware edge node grouping with fault tolerance for Internet of Battlefield Things. In Proceedings of the 2020 11th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 21–23 October 2020; pp. 420–423.
23. Raghuvanshi, K.K.; Agarwal, A.; Jain, K.; Singh, V.B. A time-variant fault detection software reliability model. *SN Appl. Sci.* **2021**, *3*, 18. [CrossRef]
24. Mavromatis, A.; Colman-Meixner, C.; Silva, A.P.; Vasilakos, X.; Nejabati, R.; Simeonidou, D. A software-defined IoT device man-agement framework for edge and cloud computing. *IEEE Internet Things J.* **2020**, *7*, 1718–1735. [CrossRef]
25. Meneghello, F.; Calore, M.; Zucchetto, D.; Polese, M.; Zanella, A. IoT: Internet of Threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet Things J.* **2019**, *6*, 8182–8201. [CrossRef]
26. Jeong, T.; Chung, J.; Hong, J.W.K.; Ha, S. Towards a distributed computing framework for fog. In Proceedings of the 2017 IEEE Fog World Congress (FWC), Santa Clara, CA, USA, 30 October–1 November 2017; pp. 1–6.
27. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I. Towards fault tolerant fog computing for IoT-based smart city applications. In Proceedings of the 2019 9th IEEE Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 752–757.